**JAVA LABSHEET 02**

**PART 01:**

1. Create a new class called ‘Item’ with two protected instance variables (private variables), an integer variable called ‘location’, and a String variable called ‘description’.

public class Item {

protected int location;

protected String description;

// Constructor

public Item(int location, String description) {

this.location = location;

this.description = description;

}

// Getters and Setters

public int getLocation() {

return location;

}

public void setLocation(int location) {

this.location = location;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

}

1. Add a constructor method for the Item class that takes an integer and a String as arguments (in that order).

public class Item {

protected int location;

protected String description;

// Constructor with arguments

public Item(int location, String description) {

this.location = location;

this.description = description;

}

// Getters and Setters

public int getLocation() {

return location;

}

public void setLocation(int location) {

this.location = location;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

}

public class Main {

public static void main(String[] args) {

// Creating an Item object using the constructor with arguments

Item item = new Item(123, "Example Item");

// Accessing and printing the properties of the Item object

System.out.println("Location: " + item.getLocation());

System.out.println("Description: " + item.getDescription());

}

}

Location: 123

Description: Example Item

1. The constructor should assign the value of these parameters to the corresponding instance variables.

public class Item {

protected int location;

protected String description;

// Constructor with arguments

public Item(int location, String description) {

this.location = location;

this.description = description;

}

// Getters and Setters

public int getLocation() {

return location;

}

public void setLocation(int location) {

this.location = location;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

}

public class Main {

public static void main(String[] args) {

// Creating an Item object using the constructor with arguments

Item item = new Item(123, "Example Item");

// Accessing and printing the properties of the Item object

System.out.println("Location: " + item.getLocation());

System.out.println("Description: " + item.getDescription());

}

}

Location: 123

Description: Example Item

1. Add getter and setter methods for the location and description variables.

public class Item {

protected int location;

protected String description;

// Constructor with arguments

public Item(int location, String description) {

this.location = location;

this.description = description;

}

// Getter for location

public int getLocation() {

return location;

}

// Setter for location

public void setLocation(int location) {

this.location = location;

}

// Getter for description

public String getDescription() {

return description;

}

// Setter for description

public void setDescription(String description) {

this.description = description;

}

}

public class Main {

public static void main(String[] args) {

Item item = new Item(123, "Example Item");

// Using getter methods

int itemLocation = item.getLocation();

String itemDescription = item.getDescription();

System.out.println("Location: " + itemLocation);

System.out.println("Description: " + itemDescription);

// Using setter methods

item.setLocation(456);

item.setDescription("New Description");

System.out.println("Updated Location: " + item.getLocation());

System.out.println("Updated Description: " + item.getDescription());

}

}

Location: 123

Description: Example Item

Updated Location: 456

Updated Description: New Description

1. Add another class called Monster and make the Monster class a sub-class of the Item class.

// Item class

public class Item {

protected int location;

protected String description;

// Constructor with arguments

public Item(int location, String description) {

this.location = location;

this.description = description;

}

// Getter and Setter methods for location and description

// (Same as previously defined)

}

// Monster class (Subclass of Item)

public class Monster extends Item {

private int strength;

// Constructor with arguments (including the arguments from the superclass)

public Monster(int location, String description, int strength) {

super(location, description); // Call the constructor of the superclass (Item)

this.strength = strength;

}

// Getter and Setter for strength

public int getStrength() {

return strength;

}

public void setStrength(int strength) {

this.strength = strength;

}

}

public class Main {

public static void main(String[] args) {

// Creating an Item object

Item item = new Item(123, "Example Item");

System.out.println("Item Description: " + item.getDescription());

// Creating a Monster object

Monster monster = new Monster(456, "Scary Monster", 100);

System.out.println("Monster Description: " + monster.getDescription());

System.out.println("Monster Strength: " + monster.getStrength());

}

}

Item Description: Example Item

Monster Description: Scary Monster

Monster Strength: 100

1. Add a constructor method to the Monster class that takes an integer and a String argument just like the Item class constructor.

// Item class (unchanged)

public class Item {

protected int location;

protected String description;

// Constructor with arguments

public Item(int location, String description) {

this.location = location;

this.description = description;

}

// Getter and Setter methods for location and description

// (Same as previously defined)

}

// Monster class (Subclass of Item)

public class Monster extends Item {

private int strength;

// Constructor with arguments (including the arguments from the superclass)

public Monster(int location, String description, int strength) {

super(location, description); // Call the constructor of the superclass (Item)

this.strength = strength;

}

// Getter and Setter for strength

public int getStrength() {

return strength;

}

public void setStrength(int strength) {

this.strength = strength;

}

}

public class Main {

public static void main(String[] args) {

// Creating a Monster object using the new constructor

Monster monster = new Monster(789, "Dangerous Monster", 200);

// Accessing and printing the properties of the Monster object

System.out.println("Monster Location: " + monster.getLocation());

System.out.println("Monster Description: " + monster.getDescription());

System.out.println("Monster Strength: " + monster.getStrength());

}

}

Monster Location: 789

Monster Description: Dangerous Monster

Monster Strength: 200

1. Use these arguments to call the Item super class constructor from within the Monster class constructor so that the instance variables in the superclass are instantiated correctly.

// Item class (unchanged)

public class Item {

protected int location;

protected String description;

// Constructor with arguments

public Item(int location, String description) {

this.location = location;

this.description = description;

}

// Getter and Setter methods for location and description

// (Same as previously defined)

}

// Monster class (Subclass of Item)

public class Monster extends Item {

private int strength;

// Constructor with arguments (including the arguments from the superclass)

public Monster(int location, String description, int strength) {

super(location, description); // Call the constructor of the superclass (Item) with the provided arguments

this.strength = strength;

}

// Getter and Setter for strength

public int getStrength() {

return strength;

}

public void setStrength(int strength) {

this.strength = strength;

}

}

public class Main {

public static void main(String[] args) {

// Creating a Monster object using the new constructor

Monster monster = new Monster(789, "Dangerous Monster", 200);

// Accessing and printing the properties of the Monster object

System.out.println("Monster Location: " + monster.getLocation());

System.out.println("Monster Description: " + monster.getDescription());

System.out.println("Monster Strength: " + monster.getStrength());

}

}

Monster Location: 789

Monster Description: Dangerous Monster

Monster Strength: 200

**PART 02**

1. Which of these keywords is used to refer to member of base class from a sub class?  
 a) upper b) super c) this d) None of the mentioned

b) **super**

The **super** keyword is used to refer to a member of the base class (superclass) from a subclass. It is commonly used in Java to access or call the superclass's constructor, methods, or instance variables. By using **super**, you can differentiate between the members of the subclass and the members of the superclass with the same name, and you can access or invoke the superclass behavior from within the subclass.

1. The modifier which specifies that the member can only be accessed in its own class is a) public b) private c) protected d) none

b) private

The **private** access modifier specifies that the member (variable or method) can only be accessed within its own class. It restricts access to that member from other classes, including subclasses (i.e., it's not inherited by subclasses). This helps to enforce encapsulation and ensures that the member is only accessible and modifiable within the class where it is declared.

1. Which of these is a mechanism for naming and visibility control of a class and its content?  
   a) Object b) Packages  
   c) Interfaces d) None of the Mentioned.

b) Packages

Packages in Java provide a mechanism for naming and visibility control of a class and its content. A package is a way to organize classes and interfaces into a hierarchical structure, and it helps in avoiding naming conflicts by providing a namespace for classes. Packages also control the visibility of classes and their members by using access modifiers like **public**, **protected**, **private**, and default (package-private). This allows for better organization, management, and access control of Java code.

1. Which of the following is correct way of importing an entire package ‘pkg’?  
   a) import pkg. b) Import pkg.  
   c) import pkg.\* d) Import pkg.\*

c) **import pkg.\***

The correct way to import an entire package **pkg** in Java is by using the statement **import pkg.\***. This statement allows you to access all the classes and interfaces within the **pkg** package without having to specify each class/interface individually. The **.\*** denotes a wildcard, indicating that all classes/interfaces within the specified package should be accessible in the current class.

For example:

import pkg.\*;

This statement will import all the classes and interfaces from the package **pkg**, allowing you to use them directly in the current Java class without specifying the full package name each time.

1. Which of these method of class String is used to extract a single character from a String object?  
   a) CHARAT() b) charat()  
   c) charAt() d) CharAt()

c) **charAt()**

The **charAt()** method of the **String** class is used to extract a single character from a **String** object. It returns the character at the specified index within the string. The index is zero-based, so the first character is at index 0, the second character is at index 1, and so on.

For example:

String str = "Hello"; char firstChar = str.charAt(0); // Gets the first character 'H' char thirdChar = str.charAt(2); // Gets the third character 'l'

The correct spelling of the method is **charAt()**, all in lowercase. It's important to use the correct spelling and casing when calling Java methods, as Java is case-sensitive.

1. Which of these method of class String is used to obtain length of String object?  
   a) get() b) Sizeof()  
   c) lengthof() d) length()

d) **length()**

The **length()** method of the **String** class is used to obtain the length of a **String** object. It returns the number of characters in the string. The length of a string is the total count of characters in the string, including spaces and special characters.

For example:

String str = "Hello"; int length = str.length(); // length will be 5

The correct spelling of the method is **length()**, all in lowercase. It's important to use the correct spelling and casing when calling Java methods, as Java is case-sensitive.

**PART 03: Fill in the blanks using appropriate term.**

1. Real-world objects contain \_\_\_ and \_\_\_.
2. A software object's state is stored in \_\_\_.
3. A software object's behavior is exposed through \_\_\_.
4. Hiding internal data from the outside world, and accessing it only through publicly exposed methods is known as data \_\_\_.
5. A blueprint for a software object is called a \_\_\_.
6. Common behavior can be defined in a \_\_\_ and inherited into a \_\_\_ using the \_\_\_ keyword.
7. A collection of methods with no implementation is called an \_\_\_.
8. A namespace that organizes classes and interfaces by functionality is called a \_\_\_.
9. The term API stands for \_\_\_?
10. **Real-world objects contain state and behavior.**
11. **A software object's state is stored in instance variables.**
12. **A software object's behavior is exposed through methods.**
13. **Hiding internal data from the outside world, and accessing it only through publicly exposed methods is known as data encapsulation.**
14. **A blueprint for a software object is called a class.**
15. **Common behavior can be defined in a superclass and inherited into a subclass using the extends keyword.**
16. **A collection of methods with no implementation is called an interface.**
17. **A namespace that organizes classes and interfaces by functionality is called a package.**
18. **The term API stands for Application Programming Interface.**